The Power of Raytracing and the Future of the Standard Shading Model

Nikita Filatov

Advanced Writing

11/23/2014

ABSTRACT

This paper reviews the applications and successes of Raytracing as a rendering algorithm, and emphasizes the superior qualities of this technique in contrast with traditional rendering methods. As game development technologies become more powerful, in pair with the growing trend of hardware becoming more capable of supporting such advancements, real time rendering is becoming the preferred method of simulating light in rendering solutions. This algorithm is also employed in simulations of particles in volumetric computations, and various applications in the medicine. This paper presents a history of the raytracing algorithm in regards to three dimensional rendering techniques, present how raytracing is the ideal form of rendering, and posit that in the near-future, as technologies develop, raytracing will be the preferred form of real time rendering.

INTRODUCTION

Raytracing has roots as far back as the 1960's, being used in military applications to predict flight paths of flying objects [19], and even for medical application like x-rays. Aside from these applications, raytracing was not widely used as an algorithm for rendering three-dimensional objects, nor was it used for physics applications like volumetric particle simulation. These are examples of hardware being a limiter on an algorithm that clearly fills the gap between reality and simulation. As video games started becoming a popular form of entertainment, it became more realistic to explore algorithms and techniques such as raycasting in that field. Games such as Castle Wolfenstein were created using raycasting alone. Since then, raytracing has become the go-to technique for almost every artist using any animation package such as Autodesk Maya or Zbrush to perform a rendering of a 3d scene.

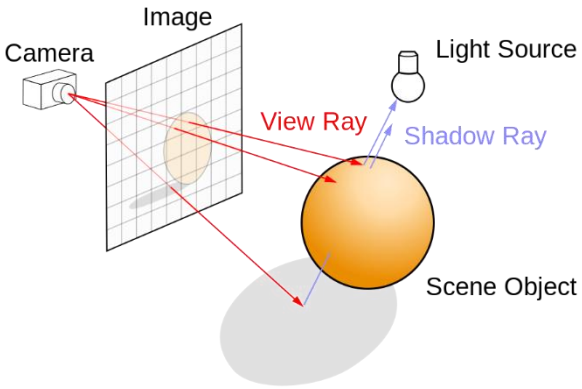

Figure 1: A visual representation of the raytracing algorithm, with 1 light and a sphere for scene geometry. [21]

IMPLEMENTATION AND BEHAVIOR

The algorithm itself is simple but expensive. It is a natural way to recreate our world, as it follows a simplified behavior of light. In order to render an image, the raytracing algorithm is performed by taking the camera's position in three-dimensional space, and casting a single mathematical ray from that point, to every pixel in the desired image. A good way to visualize this is by imagining a group of 3d objects that you wish to render, placed between a camera and a plane that is the width and height of the output image. The ray travels from the camera's position to its destination pixel on

the plane. The ray can intersect with any geometry in the scene, and behave as we would expect light rays to behave [Figure 1].

The ray computes the material properties of the collision to determine its behavior: to stop, bounce, or continue traveling. Afterwards, the ray may change its vector direction to mimic the reflective and refractive properties of the material off of which it bounced, or traveled through. Extensions of the algorithm include applications such as Subsurface Scattering, where the ray passes through a material, bounces around underneath the material's surface, and exits with a new color, among other properties. This allows for the representation of lifelike materials such as skin and marble.

The mathematical basis of the ray's behavior is a result of outward facing normal vectors on the surface of scene geometry. Once the ray strikes the surface of another piece of geometry in the scene, the renderer computes a dot product of ray's vector direction, the normal vector of the collision point, and any lights in the scene. Taking this dot product and combining it with the cosine falloff gradient of visible light produces a shading pattern that closely resembles reality [17].

Raytracing allows for light bounces and penetration of semi-transparent or fully transparent objects, but this process is recursive and therefore computationally taxing. The only thing holding back the algorithm from perfectly creating the scene, is time, which in turn is constrained by current hardware. It is not realistic to expect anyone to be able to perform fast real time raytracing, but with improvements in technology in the near future, such development are on the horizon. An example of a technological development incorporating raytracing the Optix Graphics card, which support proprietary on-board raytracing as the default hardware renderer [18].

Figure 2: A beautifully rendered scene presenting some materials which can only be created with raytracing, such as glass, plastic, ceramic, ice, and semi-transparent colored liquid. [20]

COMPARISON TO CURRENT MODELS

The behavior of light, reflective and refractive surfaces and transparency, have been hot topics in the world of 3d rendering. These few qualities are enough to describe almost any material in our world, if combined with extensions of the raytracing algorithm. Reality produces infinite variations in textures and materials, and common shading techniques do not adequately describe these materials. Raytracing is a necessity for rendering reflective surfaces such as metal and ceramic, as well as refractive surfaces like water and glass, and even translucent materials such as skin or wax paper. However, the quality of the rendering

algorithm dictates how accurately these qualia are reproduced [Figure 2].

Current shading models use the Blinn, Phong and Anisotropic material models, among many others, to create shaders in computer graphics. These shaders are primitive at best, and cannot fully describe a material. In standalone rendering, such as that used in film and animation studios, these materials are used in combination with raytracing to achieve the desired photorealistic effect. Currently, designers must overcome many obstacles in achieving photorealistic real-time rendering. The first vertex shaders introduced into computer graphics made direct use of buffered geometry to render it. That is, each vertex, edge and face was used to calculate rough shading angles to give the object a correct representation of depth based on surrounding lights, and simple specular reflection was added to give objects the appearance of glossiness.

In the development of 3d computer games, the use of normal maps was introduced. A normal map takes the traditional vertex shader, and makes use of UV texture coordinates to project a 2 dimensional mapping of the object's desired normal on a pixel perfect level, and is only accomplishable with pixel shaders. Now, where each vertex only had 1 normal, every pixel that is rendered on a face between any numbers of vertices can be projected onto a normal map to achieve incredible levels of detail without sacrificing computing power. Using normal maps gives the raytracing algorithm massive performance gains, and allows for rendering of scenes beyond what traditional shaders can accomplish, because of raytracing's many other uses, such as reflection and refraction. Adding normal maps to a reflective or refractive surface allows the algorithm to compute far more detail without the need to create complicated geometry.

ADVANTAGES OF RAYTRACING

Perfect reproduction of transparency, translucency, reflection and refraction alone account for many of the materials we see on a daily basis. The premise of accurate reflections and refractions stems from the implicit behavior of rays, in that they will follow the bending patterns of refracting light and traverse the scene in this way to collide with as many objects as the hardware permits, resulting in seemingly endless reflections, allowing us to recreate complex translucent materials such as honey [9]. During this rendering process, other natural artifacts and physical qualities of light can be introduced, such as reflection depth, blurring and inter-reflection. These qualities cannot be introduced into standard shading models because the premise of those models does not utilize rays at all. Shadows are another component of rendering that is necessary for photorealism, which traditional shading models do not support well, but raytracing supports implicitly. Another aspect of raytracing, following the behavior of light rays, is the ability for the lighting to bounce. By being able to penetrate translucent materials and bounce, colors and other qualia of a one material can be observed on adjacent materials, known as light diffusion, or global illumination [16]. This allows for the bleeding of colors between different materials, and can lead to emergent results such as water caustics. Although this process is currently time consuming and computationally demanding, some versions of the algorithm have been adapted to handle the rendering of real-time

global illumination, although at lower resolutions [13]. A modern portrayal of this system is evident in next generation game engines such as Unreal Engine 4, where global illumination is used as part of the lightmap for a scene. A lightmap is an image produced when a raytracing algorithm computes the shadows and light diffusion for static geometry in a scene. Much like a normal map, this output provides complex details without sacrificing computing power.

DRAWBACKS OF RAYTRACING AND FUTURE IMPROVEMENTS

Although the performance of real-time raytracing is undeniably sub-par when compared to standard real-time shading models, there are numerous tweaks, at the software and hardware level, which allow raytracing to be feasible real-time. Examples of performance improvements include the use of KD-trees, which reduce memory requirements by efficiently organizing the rays relative to already computed rays [2]. Another effective method of conserving memory is loading pre-fetched data into the cache [3]. Newer advancements in software and development pipelines such as storing pre-computed results of raytraced scenes in spheremaps, which are then used by the materials in the scene, allow for faster computation and rendering times as well [4]. Lastly, even though software real-time raytracing is clearly less efficient than traditional shading models, advancements in hardware, including raytracing on the CPU, and raycasting dedicated graphics cards, will soon allow for the development of feasible real-time rendering systems. [14].

CONCLUSION

Moore's law demonstrates that technology grows at an accelerating rate, and the computer graphics world has shown that people are infinitely curious about exploring photorealism, and these two aspects of the community propel each other into the future. Recent advancements in hardware and software reveal that raytracing is not only plausible in real-time, but feasible at moderate frame rates and resolutions [8]. An example of how raytracing appeared completely implausible at first, but crept into the game development pipeline is through the use of lightmap baking and shadow baking [1]. These techniques were not used at all, but now are used to generate static scene geometry. This shows how as technology grows, these tools will migrate into the real-time development pipeline. It is clear through the photorealistic representation of materials that raytracing is the superior form of rendering, and even though it seems that real-time rendering is in the distant future, physically-based rendering systems will be viable within a few years [7]. The advantage of using such an emerging technology is the introduction of new pipelines and developmental processes that use real-time raytracing as their foundation, which will lead to the creation of new languages and software [5].

Works Cited

1. Peter McGuinness, Ray tracing: the future is now, ImaginationTechnologies, June 10 2014, http://www.embedded.com/design/real-world-applications/4430971/Ray-tracing--the-future-is-now
2. B. Choi, B. Chang, I. Ihm, Improving Memory Space Efficiency of Kd-tree for Real-time Ray Tracing, Wiley Online Library, November 25 2013, http://onlinelibrary.wiley.com.ezproxy.neu.edu/doi/10.1111/cgf.12241/abstract
3. Jeong-soo Park, Woo-chan Park, Jae-Ho Nah, Tack-don Han, Node pre-fetching architecture for real-time ray tracing, IEICE Electronics Express, July 9 2013, https://www.jstage.jst.go.jp/article/elex/10/14/10_10.20130468/_article
4. Ruff, Clua, Fernandes, Dynamic Per Object Ray Caching Textures for Real-Time Ray Tracing, IEEE Xplore, 5-8 August 2013, http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6656194&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6656194
5. Bikker, Jacco, Real-time Ray Tracing through the Eyes of a Game Developer, IEEE Xplore, 2007, http://ieeexplore.ieee.org.ezproxy.neu.edu/xpl/articleDetails.jsp?arnumber=4342584
6. Marco Antonio Simon Dal Poz, Bruno Barberi Gnecco, Marcelo Knorich Zuffo, A High Performance Processor for Real-Time Ray-Tracing Image Rendering, IEEE Xplore, http://ieeexplore.ieee.org.ezproxy.neu.edu/stamp/stamp.jsp?tp=&arnumber=1594239
7. Bikker, J, Ray Tracing for Real-time Games, DataCite Content Service, http://onesearch.northeastern.edu/primo_library/libweb/action/display.do?tabs=viewOnlineTab&ct=display&fn=search&doc=TN_datacite1504024&indx=36&recIds=TN_datacite1504024&recIdxs=5&elementId=5&renderMode=poppedOut&displayMode=full&frbrVersion=2&dscnt=0&scp.scps=scope%3A%28NEU%29%2Cprimo_central_multiple_fe&frbg=&tab=default_tab&dstmp=1415757320860&srt=rank&mode=Basic&dum=true&vl(freeText0)=realtime%20raytracing&vid=NU
8. Ingo Wald, Timothy J.Purcell, Jörg Schmittler, Carsten Benthin, Philipp Slusallek, Realtime Ray Tracing and its use for Interactive Global Illumination, Eurographics 2003, https://graphics.stanford.edu/papers/egSTAR03/star03.pdf
9. Allen Ruilova, Creating Realistic CG Honey, Dreamworks Animation, https://www.cct.lsu.edu/~fharhad/ganbatte/siggraph2007/CD1/content/posters/0392.pdf
10. A. Formella, K. Muller, A viewpoint-Dependent Approach to Ray Trace free-form Surfaces, Wiley Library Online, July 7 2004, http://onlinelibrary.wiley.com.ezproxy.neu.edu/doi/10.1111/j.1467-8659.2004.00748.x/abstract
11. Mark Hachman, Caustic Announces Raytracing Card, ExtremeTech, April 20 2009, http://www.extremetech.com/computing/83048-caustic-announces-raytracing-card
12. Carsten Benthin, Realtime Ray Tracing on current CPU Architectures, Computer Graphics Group Saarland University, January 30 2006, http://scidok.sulb.uni-saarland.de/volltexte/2006/854/pdf/Dissertation_7728_Bent_Cars_2006.pdf
13. Toshiya Hachisuka, Ray Tracing on Graphics Hardware, University of California, http://www.ci.i.u-tokyo.ac.jp/~hachisuka/re.pdf

14. Kronander, Joel; Dahlin, Johan; Jonsson, Daniel; Kok, Manon; Schon, Thomas; Unger, Jonas, Real-time video based lighting using GPU raytracing, Proceedings of the 22nd European Signal Processing Conference (EUSIPCO), 2014, http://onesearch.northeastern.edu/primo_library/libweb/action/display.do?tabs=viewOnlineTab&ct=display&fn=search&doc=TN_swepubliu-107638&indx=26&recIds=TN_swepubliu-107638&recIdxs=5&elementId=5&renderMode=poppedOut&displayMode=full&frbrVersion=30&dscnt=0&scp.scps=scope%3A%28NEU%29%2Cprimo_central_multiple_fe&frbg=&tab=default_tab&dstmp=1415756533219&srt=rank&mode=Basic&dum=true&vl(freeText0)=raytracing&vid=NU
15. De Rousiers, Charles; INRIA Rhone-Alpers, Grenoble; Bousseau, Adrien; Subr, Kartic; Holzschuch, Nicolas, Real-Time Rendering of Rough Refraction, IEEE Xplore, http://ieeexplore.ieee.org.ezproxy.neu.edu/xpl/articleDetails.jsp?arnumber=6095546
16. Rochester Institute of Technology, Ray Tracing Basics, http://www.cs.rit.edu/~jmg/courses/cgII/20033/slides/3-1-raytraceBasics1.pdf
17. Andrew D. Britton, Full CUDA Implementation of GPGPU Recrusive Ray-Tracing, Purdue University, http://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=1024&context=techmasters
18. Early Raytracing History, BRL-CAD, http://brlcad.org/wiki/Early_Raytracing_History
19. Gilles Tran, Ray tracing can create realistic images, April 19 2006, http://upload.wikimedia.org/wikipedia/commons/e/ec/Glasses_800_edit.png
20. Henrik, The ray tracing algorithm builds an image by extending rays into a scene, April 12 2008,http://en.wikipedia.org/wiki/Ray_tracing_(graphics)#mediaviewer/File:Ray_trace_diagram.svg